UCRL-WEB-200720

HTAR Reference Manual

Table of Contents

Preface

Scope:	This HTAR Reference Manual explains the roles, usage, options, and features of LC's locally developed file-bundling and storage utility (the "HPSS Tape Archiver" or HTAR). HTAR is specifically designed to very efficiently transfer a very large number of (related) files as a manageable archive or library (to storage by default, or elsewhere). Besides introducing the HTAR execute line, control options, and environment variables, the text also compares HTAR with standard TAR and provides annotated examples of using HTAR for the special tasks it is designed to handle (such as retrieving files from within a stored archive, successfully managing very large archives, or depositing an archive in a designated nonstorage location).
Availability:	HTAR runs on all LC production IBM (AIX) and Linux/CHAOS machines, on both open and secure graphics machines, and on many LC special-purpose Suns.
Consultant:	For help contact the LC customer service and support hotline at 925-422-4531 (open e-mail: lc-hotline@llnl.gov, SCF e-mail: lc-hotline@pop.llnl.gov).
Printing:	The print file for this document can be found at:

```
OCF: https://computing.llnl.gov/LCdocs/htar/htar.pdf
SCF: http://www.llnl.gov/LCdocs/htar/htar_scf.pdf
```

Introduction

HTAR's Role

GOALS.
HTAR ("HPSS Tape Archiver") is an LC-designed TAR-like utility program that makes TAR-compatible archive (library) files but with storage support and enhanced archive-management features. Despite its misleading name, HTAR does *not* write files to tape, but to LC's open or secure archival storage (HPSS) disk farm (or, optionally, to other specified LC hosts).

HTAR's enhancements include its ability to:

- bundle many small files together in memory (without using more local disk space, as standard TAR requires) for more efficient handling and transfer,
- send the resulting large archive file directly to (open or secure) storage without your needing to invoke FTP separately (or to another LC machine if you use -F),
- retrieve individual files from a stored archive *without* moving the whole large archive back to your local machine first (or, optionally, without even staging the whole archive to disk), and
- accelerate transfers to and from storage by deploying multiple threads and by automatically using as many parallel interfaces to storage as are available on the (production) machine where it runs.
- easily create incremental backup archives to supplement a master archive with (only) files recently changed (with -n).

SCOPE.
A subsection of this introduction (below (page 8)) compares traditional UNIX TAR with LC's enhanced HTAR feature by feature to reveal the value of this added tool. But in general HTAR maintains full output compatibility with the POSIX 1003.1 standard TAR format while successfully archiving hundreds or even thousands of incoming files, handling files of greatly mixed sizes or types, and imposing no limit on the total size of the archive files that it builds.

PERFORMANCE.
HTAR is designed to run quickly, especially when transferring large archives to storage. HTAR does *not* invoke the separate parallel FTP (PFTP) client, but uses its own logic to efficiently manage parallel transfers to and from remote archive files (on LC machines). Its use of multiple concurrent threads and HPSS parallel disk striping enable HTAR to achieve the maximum transfer rates to storage. In most cases, creating a stored archive directly using HTAR will be much faster than either creating a local TAR file and then copying it to storage with HSI or piping TAR output into an FTP connection to HPSS or into HSI.

FILE MANAGEMENT.
HTAR can store archive-member files as large as 68 Gbyte. There is no maximum size for a whole HTAR archive other than site-imposed restrictions or amount of space available. HTAR makes single copies of each stored archive by default, but you can request dual-copy storage (for extra safety) of a mission critical archive of any size by using HTAR's -Y dualcopy option. (The HTAR -Y dualcopy option can also be specified to pick the COS for either the archive file, the index file, or both, or to specify automatic COS selection.) NFT's DIR -h command-with-option combination reveals the COS value of stored files (in output column 3).

THIS GUIDE.

Because HTAR combines two features usually separate (file bundling and file storage), having some understanding of how it works can help you use it effectively. So one subsection below shows and explains the relationship among the three files (the archive file, the index file, and the consistency file) that HTAR uses to manage every transaction. Also shown is a feature-by-feature comparison of HTAR with traditional UNIX TAR.

One section (page 10) in this manual tells how to run HTAR, and spells out common error conditions, known limitations (with work-arounds), and HTAR environment variables. A second section (page 22) describes the function of each HTAR option (distinguishing the required action options from the control options). A third section (page 33) gives annotated step-by-step examples of how to use HTAR to handle common file-archiving tasks and problems (including creation of archives containing *very* many files and optional transfer of archives to or from *non*storage LC machines).

HTAR users may also benefit from familiarity with another LC-developed specialty tool that provides nonstandard file-handling and file-transfer features linked to file storage, namely NFT (see the NFT Reference Manual (URL: https://computing.llnl.gov/LCdocs/nft) for details). HTAR itself does not, however, use NFT's "persistence" mechanisms to manage file-storage delays. For a general introduction to LC storage tools and techniques, see EZSTORAGE (URL: https://computing.llnl.gov/LCdocs/ezstorage). Hopper (page 21), LC's graphical file-handling interface, can also serve as a front end for HTAR in situations where Hopper's scalability limits are not too severe. Also of interest to the HTAR user is the HSI utility, which provides a user-friendly UNIX-style interface to storage. HSI can recursively store, retrieve, and list entire trees with a single command. Consult the HSI manual (URL: https://computing.llnl.gov/LCdocs/hsi) for details.

How HTAR Works

HTAR makes an archive (or library) file in the standard POSIX 1003.1 TAR format, which allows TAR to open any HTAR archive file. But HTAR offers more services than ordinary TAR, and it therefore needs extra internal machinery to support those services. While much of this extra machinery is hidden during normal use, some of it reveals itself in HTAR status messages or command responses that might prove surprising or confusing without some insight into how HTAR works. So this section briefly explains how HTAR makes an archive file and the role that several support files play in that process.

Archive File (*name*.tar)

When you run HTAR with the create-archive (-c) option, the program first opens a connection to storage (HPSS). It then deploys multiple threads to transfer in parallel (but not with PFTP) the local disk files that you specify (the "archive members" as illustrated in the top part of Figure 1) into a TAR-format envelope file created (unless you request otherwise) in your storage home directory (illustrated in the bottom part of Figure 1). This archive file never exists on local disk (unless you demand it with the -E option), even in temporary directories on the machine where HTAR runs. Instead, HTAR reads the member files piecewise into its internal buffers and moves the data directly to HPSS (or to another host specified by -F), where it assembles the archive (shown as archive.tar in Figure 1).

HTAR simultaneously builds a separate index file (outside the archive) and a little consistency file (deposited last inside the archive), discussed below. HPSS is very reliable, but HTAR automatically uses a storage class of service (COS) that keeps only one copy of your stored archive file. For files of special importance (only), use HTAR's -Y dualcopy option to force creation of a duplicate (invisible) backup copy. Use -K to verify your archived results.

Index File (*name*.tar.idx)

To allow archives of unlimited size and to support the direct extraction of any stored archive member(s) *without* retrieving the whole archive to local disk, HTAR automatically builds an external index file to accompany every archive that you create. While making the archive, HTAR temporarily writes the index file to the local /tmp file system on the machine where it runs, then transfers it (by default) to the same storage (or other remote) directory where the archive itself resides at the end of the process.

Each HTAR index file contains one 512-byte record for every member file, directory entry, or symbolic link stored in the corresponding archive file, regardless of the member file's size (so even a 10,000-file archive will have an index file of only about 5 Mbyte). HTAR index files are so much smaller than the archives that they support that the index file often remains on HPSS disk (to rapidly respond to queries) even when the larger archive file itself migrates to storage tape (cartridges). If you use HTAR's -E or -F options to force the archive to a location other than storage, the index file is written to the same location as the archive file.

Consistency File (/usr/tmp/HTAR_CF_CHK_*nnnnn_mmmmmmmmm*)

Because the archive and index files are separate, HTAR maintains a consistency check between them in an additional 1-block (256-byte) file always parked (as a last step) at the end of each archive. This consistency file's name has the long numerical format shown above, but it begins with /var/tmp/*uname* on a Linux/CHAOS cluster and /tmp if generated on a Sun. HTAR never extracts this file (unless you specifically request it), but every use of -t and -v (together with -c or -x) reports this perhaps unexpected consistency file at the end of HTAR's list of archived contents. (Verification option -K neither reports this consistency file nor counts it, unlike table-of-contents option -t.)

TAR and HTAR Compared

TAR was originally intended to write a specified set of files to offline tape (or retreive them), and hence by extension, to simply write (or retrieve) a specified set of files to a local envelope or library file for easier management. HTAR in many ways returns TAR to its roots because it is specifically designed to efficiently *store* a set of files together in HPSS or get them back, not merely to make an archive file and leave it, although you can force HTAR to do that.

This table compares the more familiar TAR features and effects with those (often enhanced) of HTAR:

Feature	TAR	HTAR
Can create an archive file without storing it?	Yes (the default)	With -E or -F
Can create an archive file without using local disk space?	No	Yes (the default)
Can store an archive file while creating it?	No, needs FTP	Yes (the default)
Can store an archive file without creating it?	No	No
Can write an archive to (offline) tape?	Yes	No (to storage disk)
Can write an archive to another machine?	No	Yes, with -F
Can read an archive from another machine?	No	Yes, with -F
Can read any TAR archive file?	Yes (the default)	Yes, if -X first
Can read any HTAR archive file?	Yes	Yes (the default)
Can extract just one file from a *stored* archive?	No	Yes (the default)
Can add file(s) to an existing archive?	Yes	No
Default target if no archive specified?	Yes (tape)	No, -f required
Treats input directories recursively?	Yes	Yes
Option -L disables directory recursion?	Yes (under AIX but not Linux)	No
Preserves original permissions on files?	No (uses UMASK)	Yes, with -p
Depends on HPSS availability to work?	No	Yes

Feature	**TAR**	**HTAR**
Archive duplicated automatically in storage?	No	Only with -Y dualcopy
Builds and needs an external index file?	No	Yes
Builds and needs a consistency check file?	No	Yes
Overwrites existing files without warning?	Yes	Yes (-w disables)
Can use standard input or output?	Yes (with -f -)	Yes (with -L, -O)
Order of options important?	Somewhat	Somewhat
Table of contents (-t) reveals what?	File names only	File names and properties
Can create and verify CRC checksums of member files?	No	Yes
Can verify contents of a newly created archive as part of creation operation?	No	Yes

How to Use HTAR

HTAR Execute Line

To run HTAR you must log on to an LC production machine where HTAR has been installed at a time when the storage system (HPSS) is up and available to users. The HTAR execute line has the general form

```
htar action archive [options] [filelist]
```

and the specific form

```
htar -c|t|x|D|K|U|X -f archivename [-BdEFhHILmMoOpPSTvVwY] [flist]
```

where exactly one action and the *archivename* are always required, while the control options and (except when using -c) the *filelist* (or *flist*) can be omitted (and the options can share a hyphen flag with the action for convenience). Here,

-c (create) opens a connection to storage, creates an archive file at the storage location (your home directory by default, *not* online) and with the name specified by -f, and transfers (copies) into the archive each file specified by *filelist* (required whenever you use -c). If *archivename* already exists, HTAR overwrites it without warning. To create a local archive file instead (the way TAR does), also use -E; to deposit it on a *non*storage host, also use -F. If *filelist* specifies any directories, HTAR includes them and all of their children recursively. Use -P with -c to automatically create all needed subdirectories along the archive pathname.

-t (table of contents) opens a connection to storage, then lists (in the order that they were stored) the files currently within the stored archive file specified by -f, along with their owner, size, permissions, and modification date (the list includes HTAR's own consistency file (page 6)). Here *filelist* defaults to * (all files in the archive), but you can specify a more restrictive subset (usually by making *filelist* a filter). Compare with -K output below.

-x (extract) opens a connection to storage, (or to another remote host specified by -F), then transfers (extracts, copies) from the stored (remote) archive file specified by -f each internal file specified by *filelist* (or all files in the archive if you omit *filelist*). If *filelist* specifies any directories, HTAR extracts them and all their children recursively. If any file already exists locally, HTAR overwrites it without warning, and it creates all new files with the same owner and group IDs (and if you use -p, with the same UNIX permissions) as they had when stored in the archive. (If you lack needed permissions, extracted files get your own user and group IDs and the local UMASK permissions; if you lack write permission then -x creates no files at all.) Note that -x works directly on the remote archive file; you never retrieve the whole archive from storage just to extract a few specified files from within it (impossible with TAR).

-D	(soft delete) opens a connection to storage, then reads the existing index file, creating a new temporary index file in the local file system and marking each of the specified member files as deleted in the new index file. It then replaces the existing index file with the new temporary copy.
-K	(verify) opens a connection to storage (or to another remote host specified by -F), verifies the index file for the archive that you specify with -f, then uses the index file to verify every entry in (member of) the archive file itself. The default responses from -K appear very quickly and overwrite, so you may only be able to read the last one ("HTAR successful," if it is). If the index file is missing for an archive, -K reports the error message "no such file *archivename*.idx." If you combine -K with -v (verbose output), HTAR lists the name of each file (that it successfully finds) in the specified archive in alphabetical order, one per line, along with the size of each in bytes and in blocks (excluding the consistency file (page 6)), then gives a total file count. Compare this output with that from -t (above). Note that on Linux (CHAOS) systems at LC, the -K option for standard TAR has an entirely different function than -K has for HTAR (equivalent to --starting-file, it specifies a start file during TAR extractions).
-U	(undelete) opens a connection to storage, then reads the existing index file, creating a new temporary index file in the local file system and unsetting the deleted flag for the specified member files in the new index file. It then replaces the existing index file with the new temporary copy.
-X	(index) opens a connection to storage (or to another remote host specified by -F), then creates an (external) index file for the existing archive file specified by -f (a stored TAR-format file by default, a local TAR-format file if you also use -E, or remote if you use -F). Using -X rescues an HTAR archive whose (stored) index file was lost, and it enables HTAR to manage an archive originally created by traditional TAR. The resulting external index file is stored if the corresponding archive is stored, but local if the archive is local (with -E). See the "How HTAR Works (page 6)" section for an explanation of HTAR index files.
-f *archivename*	(required option) specifies the archive file on which HTAR performs the -c|t|x|X|K actions described above. HTAR has no default for -f (whose argument must appear immediately after the option name). Because HTAR (normally) operates on *stored* archive files, *archivename* also locates the archive file relative to your *storage* (HPSS, not online) home directory: a simple file name here (e.g., abc.tar) resides in your storage home directory, while a relative pathname (e.g., xyz/abc.tar) specifies a subdirectory of your storage home directory (i.e., /users/u*nn*/*username*/xyz/abc.tar), the recommended practice for batch jobs. Never use tilde (~) in *archivename* because the shell expands it into your online, not your storage, home directory. HTAR's -f makes no subdirectories; you must have created them in advance (with FTP's or HSI's mkdir option) before you mention them in *archivename*. When used with -F to make or read an archive on a nonstorage machine, *archivename* should be the full pathname of the archive on the remote machine (e.g., /var/tmp/abc.tar).

filelist	specifies the input files for -c and the subset of archived files to process (for -t, -x, or -X). Omitting *filelist* for -c yields a null result and the error message "refusing to create empty archive." Omitting *filelist* for -t, -x, -X, or -K defaults to *, all files within the archive file specified by -f. Here *filelist* can include a blank-delimited list of files, UNIX file filters (metacharacters), or directory name(s) to be processed recursively.

SYNTAX ISSUES:
Traditional TAR is such an old utility (whose original use, writing bundled files to local tape drives, is seldom needed now) that syntax differences have evolved under different versions of the UNIX operating system. AIX and Linux (CHAOS at LC), for example, offer some different TAR options and use some of the same options (such as -L) for different purposes.

An itemized comparison of TAR and HTAR features appears in an earlier section (page 8) of this manual. Generally, HTAR syntax follows the more restrictive implementations of TAR. Thus with HTAR:
(1) one "action" -c|t|x|X|K is always required, but it need not come first on the HTAR execute line. However, if the first option on the command line starts without a minus sign but is an HTAR action character, it is treated as if the option did start with a minus sign. For example, the following two command lines are equivalent:

```
htar -c -v -f abc.tar *

htar cv -f abc.tar *
```

(2) the archive specifier -f is always required and it must *immediately* precede its argument (-f *archivename*), regardless of where that pair falls on the HTAR execute line,
(3) any HTAR flag character that requires an argument, such as -L *pathname*, requires that the argument immediately follow the option character, with or without preceding whitespace,
(4) all HTAR options, whatever their order, must precede the first member file name (all options must precede *flist* or any filters that take the place of *flist*)
(5) options may share the flag character (-) as long as the other rules above are also followed.

Thus these three combinations

```
htar -c -v -f abc.tar *

htar -cvf abc.tar *

htar -v -f abc.tar -c *
```

are all equivalent, acceptable HTAR execute lines.

DEFAULTS:

- Directories.
  By default, HTAR creates an archive by copying files from the online directory where you run it into a file in your storage (HPSS) home directory, and it extracts files by reversing that process. You must always specify the name of the archive file on which HTAR operates (there is never a default archive). In its reports, HTAR appends slash (/) to each directory name listed.
- File Names.
  Once you name the archive, HTAR calls the corresponding external index file *archivename*.idx by default and stores it in the same HPSS directory as the archive (by default). HTAR's -I option lets you specify an nondefault name or location for the index file. The HTAR consistency file's name begins with /usr/tmp/HTAR on IBM machines and /tmp/HTAR on Sun machines (a default that you cannot change). On Linux/CHAOS machines, the consistency file's name begins with /var/tmp/*uname*/HTAR, where *uname* is your login name on the machine where you run HTAR.

- Class of Service (COS).
  By default, HTAR stores a single copy of each archive in HPSS, regardless of its size. But you can request dual-copy storage of any mission critical HTAR archive, regardless of its size, by using the -Y dualcopy option on HTAR's execute line. NFT's command DIR used with the -h option reports the COS for stored files (in output column 3).

HTAR Error Conditions

Following LLNL practice, HTAR prefixes all ordinary messages with the string 'HTAR:', but it prefixes nonfatal errors with 'INFO:' and fatal errors with 'ERROR:'. Unexpected situations are usually flagged with a '###WARNING' prefix.

The most common error conditions and HTAR's (often cryptic) responses to them are summarized here to help you troubleshoot:

Storage (HPSS) is down.

When HPSS is unavailable to users (perhaps for maintenance), no stored archive can be read or written. HTAR returns a message of this form and ends (there is no persistence as with NFT):

```
hpssex_OpenConnection: unable to obtain remote site info
result = -5000, errno = 0
Unable to setup communication to HPSS. Exiting...
```

Specified archive directory does not exist.

If -f specifies a child directory (of your storage home directory) that you have *not* previously created (with FTP's or HSI's mkdir option), HTAR returns an error message that often only hints at the actual problem. When you attempt to create an archive in a nonexistent (sub)directory, HTAR responds:

```
***Error -2 on hpss_Open (create) for archivename
```

When you attempt to extract files from an archive in a nonexistent (sub)directory, HTAR at least replaces the first line of this error message with:

```
***Fatal error opening index file archivename.idx
```

Specified archive file does not exist.

If -f specifies an archive file that does not exist (perhaps because you deleted it or mistyped its name), HTAR responds:

```
[FATAL] no such HPSS archive file: archivename
```

Specified index file does not exist.

If you try to list (-t) or extract (-x) files from an actual HTAR archive whose corresponding external index file (*archivename*.idx) has been deleted or moved, HTAR pinpoints the problem only by reporting the missing index name:

```
No such file: archivename.idx
```

You can work around the missing index by using HTAR's -X (uppercase eks) option to rebuild the index while the archive remains stored, or you can retrieve the whole archive from storage with FTP or HSI and then open it with TAR.

HTAR's *filelist* omitted.

If you try to create (-c) an archive without specifying a *filelist* (or without using a *filelist* replacement such as -L), HTAR connects to HPSS but quickly ends with the message

```
   Refusing to create empty archive.
```

If you try to list (-t) or extract (-x) without specifying a *filelist*, HTAR defaults to processing all files in the archive.

HTAR run with no options.

Because HTAR requires exactly one action (-c|t|x|X|K) and a specified archive file (-f) to run, executing the program with nothing else on the execute line yields a terse syntax summary. There is no prompt for input, and HTAR terminates.

Command line too long for shell.

The easy way to build an HTAR archive of *very many* like-named files is to specify them indirectly by using a UNIX metacharacter (filter, wild card) such as * (to match any string) or ? (to match any single character). But if the selected file set has thousands of members, the list of input names that the UNIX shell generates by expanding such an "ambiguous file reference" may grow too long to handle. See the Limitations and Restrictions (page 17) section below for several ways to work around such excessively long command lines when building large archives with HTAR.

Wild cards (metacharacters) used for retrieval.

HTAR allows * only to create an archive, not to retrieve files from one ("no match" is the usual, but not the only possible, error message). See the Retrieving Files (page 35) section below for an analysis and possible ways to work around this limitation.

HTAR Limitations and Restrictions

The current version of HTAR has the following known limitations or usage restrictions:

- I/O:
  You can redirect any HTAR output into a file (with >) for separate postprocessing (see the Retrieving Files (page 35) section for one helpful application of this). But HTAR normally does not read from or write to UNIX pipes (standard input, standard output). Two HTAR control options, however, let you enable the use of pipes if you need them:

  - ◊ Read From Standard Input.
    Use HTAR's -L *inputfile* option with a hyphen (-) as the *inputfile* (that is, -L -) to read a list of files from a UNIX pipe (from standard input) instead of from the usual execute-line *filelist*. The "Too Many Names" discussion later in this section shows how to apply this technique to solve a practical problem when creating archives with very many input files.
  - ◊ Write To Standard Output.
    Use HTAR's -O (uppercase oh) option to write a file extracted with the -x option to standard output (and hence to a UNIX pipe if you wish). Thus

    ```
    htar -xf abc.tar -O def
    ```

    extracts file DEF from archive ABC.TAR in your storage home directory and displays it at your terminal, while

    ```
    htar -xf abc.tar -O def | wc
    ```

    instead reports DEF's line, word, and character count. Because HTAR does not separate files in the output stream, this usually is useful only when extracting a single file.

- METACHARACTERS:
  HTAR leaves all processing of metacharacters (filters or wild cards, such as *) to the shell. This means that when you create an HTAR archive you can use * to select from among your local files to store, but when you *retrieve* specific files from within an already stored archive you CANNOT use * to select from among the stored files to get back. See the Retrieving Files (page 35) example below for details on this limitation, and a few suggested but inelegant ways to work around it. Another side effect of this approach to metacharacters is that C shell (csh) users must type the three-character string -\? (instead of -?) to display HTAR's help message.
- UPDATES:
  No options exist to update (replace), remove (delete), or append to individual files that HTAR has already archived. You must replace (create again) an entire archive to alter the member files within it.
- NAME LENGTH:
  To comply with POSIX 1003.1 standards regarding TAR-file input names, the longest input file name of the form prefix/name that HTAR can accept has 154 characters in the prefix and 99 characters in the name. Link names likewise cannot exceed 99 characters.

- FILE SIZE:
  The maximum size of a single member file within an HTAR archive is 68 Gbyte (expanded from the former 8-Gbyte limit once imposed by the format of the TAR header). HTAR imposes no limit on the maximum size of an archive file (some have successfully reached 10 Tbyte), but local disk space (when using -E or -F) or storage space might externally limit an archive's size. Users can specify a maximum number of member files per archive with HTAR's -M option.
- PASSWORDLESS FTP:
  Because HTAR (unlike FTP) does not support user dialog with a server and has no password-passing option, you can only manipulate HTAR archives on machines with preauthenticated (passwordless) FTP servers. This limits the use of HTAR's -F option to LC production machines only because there is no way to satisfy the password request from other FTP servers. HTAR does *not* run the PFTP client.

TOO MANY NAMES.
For users who make HTAR archives containing thousands of files, a different kind of limitation poses problems, a limitation of the UNIX shell (csh, bsh, ksh) rather than of the HTAR program itself. One would normally select multiple files for archiving by using a UNIX "ambiguous file reference," a partial file name adjacent to one or more shell metacharacters (or "wild card" filters, such as the asterisk(*)). Your current shell automatically expands the metacharacter(s) to generate a (long) alphabetical list of matching file names, which it inserts into the execute line as if you had typed them all yourself. Thus

```
    htar -cf test.tar a*
```

might become equivalent to a command line with dozens of a-named files on the end. Each shell has a maximum length for execute lines, however, and if your specified metacharacter filter matches thousands of file names, HTAR's execute line may grow too long for the shell to accept. This would prevent building your intended many-file archive.

WORK-AROUND 1: USE A DIRECTORY.
The most effective, least resource-intensive way to work around the problem of having a (virtual) HTAR execute line too long for the shell to handle is to plan ahead and keep (or generate) in a single directory all and only the files that you want to archive. HTAR processes directory names recursively by default. So, if you specify only the relevant directory name on HTAR's execute line, HTAR will (internally) archive every file within the directory without any filter-induced length problems. For example,

```
    htar -cf test.tar projdir
```

will successfully archive any number of files within the PROJDIR directory yet use no troublesome shell-mediated file-name generation to do it.

WORK-AROUND 2: USE FIND.
The UNIX FIND utility is designed to produce lists of files (that meet specified criteria) to feed into other programs for further processing, so FIND offers a second way for HTAR to archive very large numbers of files without having a very long execute line. Indirection is required for success, however. The "natural" use of FIND's -EXEC option to run another program (here, HTAR) driven by a list of files from FIND, for example,

```
    [WRONG]
  find . -name 'a*' -print -exec htar -cf test.tar {} \;
```

fails to produce the desired effect. This actually runs HTAR once (to build an archive called test.tar) for each successive input file (here, files beginning with A). If there are thousands of files, HTAR just repeatedly creates a one-file archive thousands of times (each replacing the previous archive) so that only the last file processed really remains in test.tar at the end.

Instead, enable FIND to pipe standard input directly into HTAR by invoking HTAR's -L option with a hyphen (-) argument instead of a file name. The correct sequence is:

```
    find . -name 'a*' -print | htar -cf test.tar -L -
```

(The use of the metacharacter * in FIND's execute line here does not pose the same too-long problem as it did originally in HTAR's execute line because the surrounding quotes shelter the filter from shell processing. FIND's -NAME option generates the list of matching names internally, without expanding FIND's execute line.) If you need to keep the list of input names (for verification or audit purposes, for example), you could break this single line into two equivalent steps mediated by a helper file (here called ALIST) that you preserve:

```
      find .  -name 'a*'  -print > alist
      htar -cf test.tar -L alist
```

HTAR Environment Variables

HTAR uses the following HPSS-related environment variables if they are available on the machine where it runs:

HPSS_HOSTNAME

specifies the host name or IP address of the network interface to which HPSS mover(s) should connect when transferring data at HTAR's request (overridden by the information specified in the file /usr/local/etc/HPSS.conf). The default interface (the alternative to HPSS_HOSTNAME) is often slow, such as the control Ethernet of an IBM SP machine.

HPSS_PATH_ETC

specifies the pathname of a local directory containing the HPSS network options file.

HPSS_SERVER_HOST

specifies the server host name and optional port number of the HTAR server.

HTAR_COS specifies the default class of service (COS) ID for the archive file that HTAR creates, or contains the string AUTO to force HPSS to automatically select the class of service based on the file size. HTAR option -Y overrides HTAR_COS (see the end of the next section (page 25) for details).

TMPDIR specifies the directory to use when HTAR creates temporary files, such as the index file or the consistency file.

Executing HTAR Using Hopper

Hopper is a graphical front end for several file-transfer tools (FTP, NFT, HTAR) that is installed on all LC production machines, open and secure. With Hopper you can create HTAR archives by graphically dragging files and directories to the Hopper storage window, and you can just as easily extract contents of HTAR archives. For more details on Hopper, type "man hopper" on an LC host, use Hopper's built-in help package, or visit the Hopper Web pages at https://computing.llnl.gov/resources/hopper/ (URL: https://computing.llnl.gov/resources/hopper/).

HTAR Options

Action Options

Exactly ONE of these action options is required every time that you run HTAR.

-c	(create) opens a connection to storage (in parallel, but not using PFTP), creates an archive file at the storage location (*not* online) and with the name specified by -f, and transfers (copies) into the archive each file specified by *filelist* (required whenever you use -c). If *archivename* already exists, HTAR overwrites it without warning. To create a local archive file instead (the way TAR does), also use -E; to deposit it on a *non*storage host, also use -F. If *filelist* specifies any directories, HTAR includes them and all of their children recursively. Use -P with -c to automatically create all needed subdirectories along the archive pathname.
-D	(delete) soft-deletes the specified member files from the archive by marking them as deleted in the index file. When the archive file is repacked (planned for a future release), deleted files will not be included in the repacked archive.
-K	(verify) opens a connection to storage (or to another remote host specified by -F), verifies the index file for the archive that you specify with -f, then uses the index file to verify every entry in (member of) the archive file itself. The default responses from -K appear very quickly and overwrite, so you may only be able to read the last one ("HTAR successful," if it is). If the index file is missing for an archive, -K reports the error message "no such file *archivename*.idx." If you combine -K with -v (verbose output), HTAR lists the name of each file (that it successfully finds) in the specified archive in alphabetical order, one per line, along with the size of each in bytes and in blocks (excluding the consistency file (page 6)), then gives a total file count. Compare this output with that from -t (above). Note that on Linux (CHAOS) systems at LC, the -K option for standard TAR has an entirely different function than -K has for HTAR (equivalent to --starting-file, it specifies a start file during TAR extractions).
-t	(table of contents) opens a connection to storage (in parallel, but not using PFTP), then lists (in the order that they were stored) the files currently within the stored archive file specified by -f, along with their owner, size, permissions, and modification date (the list includes HTAR's own consistency file (page 6)). Here *filelist* defaults to * (all files in the archive), but you can specify a more restrictive subset (usually by making *filelist* a filter).
-U	(undelete) undeletes the specified member files from the archive that were previously soft-deleted by -D by removing the deleted flag in their index file entires.

-x	(extract) opens a connection to storage (or to another remote host specified by -F), then transfers (extracts, copies) from the stored (remote) archive file specified by -f each internal file specified by *filelist* (or all files in the archive if you omit *filelist*). If *filelist* specifies any directories, HTAR extracts them and all their children recursively. If any file already exists locally, HTAR overwrites it without warning, and it creates all new files with the same owner and group IDs (and if you use -p, with the same UNIX permissions) as they had when stored in the archive. (If you lack needed permissions, extracted files get your own user and group IDs and the local UMASK permissions; if you lack write permission then -x creates no files at all.) Note that -x works directly on the remote archive file; you never retrieve the whole archive from storage just to extract a few specified files from within it (impossible with TAR).
-X	(index) opens a connection to storage (or to another remote host specified by -F), then creates an (external) index file for the existing archive file specified by -f (a stored TAR-format file by default, a local TAR-format file if you also use -E, or remote non-storage machine if you use -F). Using -X rescues an HTAR archive whose (stored) index file was lost, and it enables HTAR to manage an archive originally created by traditional TAR. The resulting external index file is stored if the corresponding archive is stored, but local if the archive is local (with -E). See the "How HTAR Works (page 6)" section for an explanation of HTAR index files.

Archive Option

This option is required every time that you run HTAR unless you only use the -? option.

-f *archivename*

> (required option) specifies the archive file on which HTAR performs the -c|t|x|X|K actions described above. HTAR has no default for -f (whose argument must appear immediately after the option name). Because HTAR (normally) operates on *stored* archive files, *archivename* also locates the archive file relative to your *storage* (HPSS, not online) home directory: a simple file name here (e.g., abc.tar) resides in your storage home directory, while a relative pathname (e.g., xyz/abc.tar) specifies a subdirectory of your storage home directory (i.e., /users/u*nn*/*username*/xyz/abc.tar). Never use tilde (~) in *archivename* because the shell expands it into your online, not your storage, home directory. HTAR's -f makes no subdirectories; you must have created them in advance (with FTP's or HSI's mkdir option) before you mention them in *archivename*. When used with -F to make or read an archive on a nonstorage machine, *archivename* should be the full pathname of the archive on the remote machine (e.g., /var/tmp/abc.tar).

Control Options

These options change how HTAR behaves, but none is required (default values are indicated when they exist).

-?	displays a short help message (a syntax summary of the HTAR execute line and a one-line description of each option). Users running HTAR under the C shell (CSH) will probably have to use the three-character string -\? to display this help message.
-B	adds block numbers to the listing (-t) output (normally used only for debugging).
-d *debuglevel*	(default is 0) sets to an integer from 0 through 5 the level of debug output from HTAR, where 0 disables debug information for normal use and 1 to 5 enable progressively more elaborate debug output.
-E	emulates TAR by forcing the archive file to reside on the *local* machine (where you run HTAR) rather than in HPSS (storage), where it resides by default (-f always specifies the archive pathname, which -E interprets as local rather than remote). See also -F for making nonstorage *remote* archives. The HTAR index file goes into the same (local) directory as the archive. Option -P works with -E.
-F [*user@*]*host*[*#port*]	overrides the HTAR default of a *stored* archive and specifies on which remote machine (*host*) the archive resides other than in HPSS. For creating archives, *host* is the sink machine; for extracting files from existing archives, *host* is the source machine (see the between-machine example (page 44) for how to use -F properly). See also -E for making nonstorage *local* archives. The HTAR index file goes into the same (remote) directory as the archive. Any LC production machine with preauthenticated (passwordless) FTP service can be the -F *host*. HTAR still contacts the HPSS server even though the archive does not reside in HPSS, just to log all -F transactions. The *user* and *port* fields are seldom needed or appropriate because they usually betray the need for an FTP password and HTAR has no means to transmit one (the default *user* is you, the default *port* is 21). WARNINGS: Using -F to specify "storage" as the archive host is not only completely unnecessary, because it is HTAR's default, but also inefficient, because it undermines HTAR's built-in techniques for moving files to or from *stored* archives quickly and effectively. Also, -P does *not* work with -F.
-h	(used only with -c; has no effect otherwise) for each symbolic link that it encounters, causes HTAR to replace the link with the actual contents of the linked-to file (stored under the link name, not under the file's original name). Later use of -t or -x treats the linked-to file as if it had always been present as an actual file with the link name. Without -h, HTAR records, reports, and restores every symbolic link overtly, but it does *not* replace the link with the linked-to contents.

-H *subopt[:subopt...]*

specifies a colon-delimited list of HTAR suboptions to control program execution. Possible *subopt* values include:

acct=*id/acctname*

specifies the numeric account ID or alphabetic account name to use for the current HTAR run. This option is only meaningful for HPSS-resident archives.

cix	used with the extract (-x) operation with HPSS-resident archives. If specified, precopies the index file to a temporary local file before reading the archive file. This option is normally not needed, but was added to avoid problems that were encountered with multithreaded I/O on some hardware platforms.
crc	enables generation of Cyclic Redundancy Checksums (CRCs) when copying member files into the archive and when verifying the contents of the archive (-K command line option, or -Hverify option for creates). Enabling checksums usually degrades HTAR's I/O performance and increases its CPU utilization.
exfile=*path*	specifies a path name to an "exceptions" file, which contains a list of failed member files and an explanation of the failure. Note: This option is currently implemented only for the GPFS/HPSS Interface (GHI).

family=*id[,index_id]*

specifies tape file family ID to use when creating HPSS-resident archive files, and, optionally, the family ID to use when creating the index file. This option is useful at sites which make use of the HPSS "file family" capability. Family ID 0, which is the default, uses the default pool of tapes. Contact your HPSS administrator to determine the file families that are available at your site.

nocfchk	causes HTAR to disable the verification of the index file and the consistency file. Use of this option can avoid extra tape mounts if the consistency file lives on a different tape cartridge than the specified member file(s). Currently, this option is only effective for the -D (soft-delete) action.
nocrc	(the default) disables generation of CRCs when creating files and when extracting files from or verifying existing archive files.
nostage	avoids prestaging tape-resident (stored) archive files when HTAR performs -x or -X actions.

okfile=*path*	specifies path to a file to contain a list of successfully transferred files. Note: this option is currently implemented only for the GPFS/HPSS Interface (GHI).
port=*x*	specifies the TCP port number to use when HTAR connects to the remote HPSS server. This parameter is only used in conjunction with the -Hserver parameter.
relpaths	used with the verify (-K) action. When comparing member files in the archive file with local files, forces relative local file paths to be used by removing any leading "/" from the member file path name before attempting to read it in the local file system.
rmlocal	removes local member files after HTAR has successfully written both the archive file and the index file (used with -c).
server=*host*	specifies the hostname or TCP/IP address of the HPSS server. The HPSS administrator defines the default server host or IP address when HTAR is built. The -Hport parameter (see above) can be used in conjunction with this option to completely specify the connection address to be used.
tss=*stack_size*	specifies the thread stack size to be used when HTAR creates threads to read local files during a create (-c) operation. In most cases, the system default value can be used, but situations such as the case where the default thread stack size is set very large, for example, on machines that are tuned for compute-type problems, can cause HTAR thread creations to fail. *stack_size* can be specified in bytes, kilobytes, or megabytes by appending a case-insensitive suffix (k, kb, m, or mb).
umask=*octal_mask*	used with the -c option. This specifies the HPSS umask value to be set during HTAR startup. This impacts the permissions that are set on the resulting archive and index files that HTAR creates in the same manner as the Unix umask command.
verify=*option*[,*option*,..]	specifies one or more verification options that should be performed following successful creation of the archive (-c), or for the verify (-K) command. Multiple options can be specified by separating them with a comma, with no whitespace. Options are processed from left to right, and, in the case of conflicting options, the last one encountered is used without comment. The options can be either individual items or the keyword "all" or a numeric value of 0, 1, or 2. Each numeric level includes all of the checks for lower-valued levels and adds additional checks. The verifications options are:

all

enables all possible verification options except paranoid.

info

reads and verifies the tar-format headers that precede each member file in the archive.

crc|noncrc

enables or disables recalculation of the cyclic redundancy checksum (CRC) and verification that it matches the value that is stored in the index file. Note that this option only applies if the -Hcrc option was specified, which causes a CRC to be generated for each member file as it is copied into the archive file.

compare|nocompare

enables or disables byte-by-byte comparison of the local member files with the corresponding archive files. If -Hrelpaths is not specified, then absolute paths for member files in the archive will also be treated as absolute local paths.

paranoid|noparanoid

enables or disables (the default) extreme efforts to detect problems (such as discovering whether local files were modified during archive creation before deleting them if authorized by RMLOCAL).

0|1|2

0 enables the "info" verification. 1 enables level 0 and "crc" (i.e., info,crc). 2 enables level 1 and "compare" (i.e., info,crc,compare). It is also possible to specify a verification option such as "all" or a numeric level such as 0, 1, or 2, and then selectively disable one or more options.

-I *indexname*

specifies a nondefault name for the HTAR external index file that supports the archive specified by -f.
WARNING: if you use -I to make any nondefault index name (3 cases, below) when you create (-c) an archive, then you MUST also use -I with the same argument every time you

extract (-x) files from that archive (else HTAR will look for the default index, not find it, and end with an error).
There are three cases based on the first character of *indexname*:

. (dot)	If *indexname* begins with a period (dot), HTAR treats it as a suffix to append to the current archive name. Example: -I .xnd yields an index file called *archivename*.xnd
/	If *indexname* begins with a / (slash), HTAR treats it as an absolute path name (you must create all the subdirectories ahead of time with FTP's or HSI's mkdir option). Example: -I /users/u*nn*/*yourname*/projects/text.idx uses that absolute path name in storage (HPSS) or the local file system (-E) or remote file system (-F) for the index file.
other	If *indexname* begins with any other character, HTAR treats it as a relative pathname (relative to the storage directory where the archive file resides, which might be different than your storage home directory). Example: -I projects/first.index locates first.index at *storagehome*/ projects/first.index if the archive file is in your *storagehome* (the default), but tries to locate first.index at *storagehome*/projects/projects/ first.index if the archive was specified as -f projects/aname in the first place. (All such subdirectories must be created in advance or the -P command line option must be specified to create any missing intermediate subdirectories.)

-L *inputfile* (used with -c) writes the files and directories specified by their literal names (in the *inputfile*, which contains file names one per line) into the archive specified by -f. Directories are treated recursively; a directory entry and its subdirectories or subfiles are all written to the archive. Normal metacharacters (tilde, asterisk, question mark) are treated literally, not expanded as filters. Replace *inputfile* with a hyphen (-L -) for HTAR to read the list of file names from standard input; the HTAR "Limitations" section (page 17) shows how to use this technique.
(used with -x) retrieves the files and directories specified by their literal names. See the Retrieving Files (page 35) example below for how to use -L instead of wild cards to retrieve only specified files from a stored archive.
WARNING: HTAR's -L differs from both AIX TAR's -L (which handles directories *non*recursively) and Linux TAR's -L (which changes tapes).

-m (used only with -x; applies only to files) makes the time of extraction the last-modified time for each member file (the default preserves each file's original time of last modification). For directories, HTAR itself always preserves the original modification time for top-level directories that it copies from an archive, even if you invoke -m. However, subsequently creating subdirectories or files within a directory may cause the operating system to change the modification time on one or more directories (so that it too appears to be the time of extraction).

-M *maxfiles*	(default is 10,000,000 at LLNL) specifies the maximum number of member files allowed when you use -c to create an HTAR archive. Internal limits are set when HTAR is compiled at each site; at LLNL, you can increase *maxfiles* as high as 50,000,000.
-n *timeinterval*	(used only with -c; has no effect otherwise) includes in a new archive only those files (that meet your other naming criteria and) that were either created or modified between now and the start of *timeinterval*. Option -n is intended mostly to simplify the creation of incremental backup archives. Here *timeinterval* can have the form:

d	an integer that specifies days (e.g., 5 for 5 days), or
:h	an integer that specifies hours (e.g., :12 for :12 hours), or
d:h	a pair of integers that specify days and then hours (e.g., 1:6 for 1 day and 6 hours).

-o	(lowercase, used only with -x) (default for all nonroot users) causes the extracted files to take on the user and group ID (UID, GID) of the person running HTAR, not those of the original archive. This makes a difference for root users but not for ordinary HTAR users.
-O	(uppercase, used only with -x, mimics the Linux TAR --to-stdout option) writes the file(s) extracted from an archive (with -x) to standard output (and hence to a UNIX pipe for postprocessing, if you wish). The HTAR "Limitations" section (page 17) above shows how to use this technique. Because HTAR does not separate files in the output stream, -O is usually useful only when you extract a single file.
-p	preserves all UNIX permission fields (on extracted files) in their original modes, ignoring the present UMASK (the default changes the permissions to the local UMASK where HTAR extracts the files). Root users can also preserve the setuid, setgid, and sticky bit permissions with this option.
-P	(used only with -c, has no effect otherwise) automatically creates all intermediate subdirectories specified on the archive file's pathname if they do not already exist. HTAR's -P thus works the same as MKDIR's -P option. You can use -P with archives created in HPSS (storage, the default) or on your local machine (with -E), but it fails for other remote archives (those created with -F).
-q	(quiet mode) supresses most HTAR informational messages, such as its usual interactive progress reports as it creates an archive file.
-S *bufsize*	(default is 16 Mbyte) specifies the buffer size to use when HTAR reads from or writes to an HPSS archive file. Here *bufsize* can be a plain integer (interpreted as bytes), an integer suffixed by k, K, kb, or KB for kilobytes, or an integer suffixed by m, M, mb, or MB for megabytes (e.g., 16mb). -S is intended mostly for LC staff, not ordinary HTAR users.

-T *maxthreads*	specifies the maximum number of threads that HTAR will use to copy member files to or from the archive file (default varies from 5 to 20 threads). This value is ignored when extracting member files from an archive (-x). HTAR reports the actual number of threads used on each run if you invoke -v or -V. HTAR creates a *maxthreads* pool of threads and then uses buffer size (see -S), average member file size, and HPSS network transfer rates to estimate how many threads to actually deploy. Normally, the smaller the member file size, the more threads can be active when creating files. For small files, setting -T to a larger number (up to 100 has been tested) can dramatically improve the transfer rates if the operating system is able to support the load.
-U	undeletes soft-deleted member files (see -D above) by copying the existing index file to a temporary local file, removing the deleted flag in the specified index entries along the way, and then rewriting the temporary index to the same location (HPSS, local file system (-E) or remote filesystem (-F)).
-V	requests "slightly verbose" reporting of file-transfer progress (often very brief, overwritten messages to the terminal). Do not use with -v.
-v	requests "very verbose" reporting of file-transfer progress. For each member file transferred to an archive, HTAR prints A (added) and its name on one line; for each member file extracted from an archive, HTAR prints X, its name, and its size on a line, along with a summary of the whole transfer at the end. For each file added during a build index (-X) operation, HTAR prints i and its name. For each file verified during a verify operation (-K), HTAR prints v (or V if comparing archive and local file contents), its name, and a trailing / if this is a directory. For each file that is soft-deleted during a delete (-D) operation, HTAR prints d; similarly, for an undelete (-U) operation, HTAR prints u. Do not use with -V.
-w	(works only with -x, -D, -U, not with -c) lists (one by one) each member file to be extracted from the archive and prompts you for your choice of confirmatory action, where possible responses are:

y[es]	extracts the named file.
n[o]	skips the named file.
a[ll]	extracts the named file and all remaining (not yet processed) selected files too.
q[uit]	skips the named file and stops prompting. HTAR ends.

-Y auto | [*archiveCOS*][:*indexCOS*]

specifies the HPSS class of service (COS) for each stored archive and its corresponding index file. The default is AUTO, which causes HTAR to use a site-specific COS chosen for archive suitability (at LC, the default COS for HTAR files is 160, which automatically stores a single copy of each archive, regardless of its size). You can specify a nondefault COS for the archive, the index, or both (e.g., -Y 120:110), but this is usually undesirable except when testing new HPSS features or devices (if your archive size grows to exceed that allowed by a nondefault COS, HPSS will stop the transfer and HTAR will end with an error). Use -Y dualcopy (i.e., COS 200) to request dual-copy storage of any mission critical archive of any size for extra safety. Using -Y overrides the HTAR_COS environment variable. NFT's DIR command with the -h option reports the COS for stored files (in output column 3), while NFT's SETCOS command offers a different way to specify the storage class of service.

HTAR Examples

Creating an HTAR Archive File

GOAL: To create an HTAR archive file in a subdirectory of your storage home directory and use a filter to install several files within that stored archive.

STRATEGY: (1) One HTAR execute line can perform all of the desired tasks quickly and in parallel:

- The -cvf options create (c) an archive, verbosely (v) report the incoming files, and (f) name the envelope file.
- The relative pathname case3/myproject.tar locates the archive (myproject.tar) in pre-existing subdirectory case3 of your storage home directory (omitting case3/ leaves the archive at the top level of your storage home directory). HTAR will not create case3 by default, however; you must either have previously used FTP's or HSI's mkdir command or else you must invoke -P to explicitly request creation of all needed subdirectories along the archive pathname.
- File filter tim* selects all and only the files whose names begin with TIM (in the directory where you run HTAR) to be stored in the archive.

(2) HTAR opens a preauthenticated connection to your storage (HPSS) home directory and reports its housekeeping activities (very quickly, in lines that overwrite, so you may not notice all of these status reports on your screen).
(3) HTAR creates your requested archive and uses parallel connections (but not the PFTP client) to move your requested files directly into it. Directories are handled recursively and directory names (if any) appear with a slash (/) appended to identify them.
(4) The last incoming file that HTAR reports is always the 256-byte consistency file by which HTAR coordinates your archive with its external index file.
(5) HTAR summarizes the work done (time, rate, amount, thread count), then copies into storage the index file that it made, destroys the local version, and ends.

```
htar -cvf case3/myproject.tar tim*                        ---(1)

  HTAR: Opening HPSS server connection                    ---(2)
  HTAR: Getting HPSS site info
  HTAR: Writing temp index file to /usr/tmp/aaamva09A

  HTAR: creating HPSS Archive file case3/myproject.tar    ---(3)
  HTAR: a   tim1.txt
  HTAR: a   tim2.txt
  HTAR: a   tim2a.txt
  HTAR: a   tim3.a
  HTAR: a   time.txt
  HTAR: a   time2.gif
```

```
HTAR: a   /tmp/HTAR_CF_CHK_13805_997722535               ---(4)

HTAR: Create complete for case3/myproject.tar. 399,360   ---(5)
        bytes written for 6 member files, max threads: 8
      Transfer time: 0.555 seconds (7.257 MB/s)

HTAR: Copying Index File to HPSS...Creating file
HTAR: HTAR SUCCESSFUL
```

Retrieving Files from within an Archive

GOAL: To retrieve several files from within an existing stored HTAR archive file (without retrieving the whole archive first).

STRATEGY: HTAR does not process metacharacters (file filters such as *) itself, but leaves them for the shell to expand and compare with file names in your *local* directory. Hence, you CANNOT use * to select a subset of already archived files to retrieve. For example, "natural" execute lines

```
htar -xvf case3/myproject.tar time*     [WRONG]
htar -xvf case3/myproject.tar 'time*'   [WRONG]
```

both FAIL to select (and hence to retrieve) any stored files from the MYPROJECTS.TAR stored archive (each yields its own set of error messages). These lines work only accidentally, if you happen to have files with the same name in both your local directory and your stored archive (unlikely except when you are just testing HTAR).
WORKAROUNDS:
(1A) Type the name of each file that you want to retrieve (at the end of the HTAR execute line).
(1B) If you have a long list of files to retrieve, or if you plan to reuse the same retrieval list often, put the list of sought files into a file and use HTAR's -L option to invoke that list. You can use HTAR's -t (reporting) option to help generate that retrieval list by reporting all the files you have archived and then editing that report to include only the relevant file names to retrieve. For instance,

```
htar -tf case3/myproject.tar > hout
grep 'time' hout | cut -c 50-80 > tlist
```

captures the list of all your stored files in the local file HOUT and then selects just the file names that contain the string TIME for use with HTAR's -L option (here, in local file TLIST). Note that HTAR automatically appends slash (/) at the end of every directory name that -t reports.
(1C) Use HOPPER to run HTAR as a controllee, then select *visually* the files that you want to retrieve.

(1) Once you have laid the groundwork above, a single HTAR execute line can retrieve your specified files quickly and in parallel from within your stored archive:

- The -xvf options request retrieval/extraction (x), verbosely (v) report the retrieved files, and (f) name the target archive.

- The relative pathname case3/myproject.tar locates the archive (myproject.tar) in pre-existing subdirectory case3 of your storage home directory.
- The explicit file list (1A) or name-containing file (1B) selects all and only the files that you want (here, those whose names begin with TIME, a subset of all files stored in this archive in the previous example).

(2) HTAR opens a preauthenticated connection to your storage (HPSS) home directory and reports its housekeeping activities (very quickly, in lines that overwrite, so you may not notice all of these status reports on your screen).
(3) HTAR uses its external index to locate in the archive the (two) specific files that you requested and then it transfers them by using parallel connections (but not the PFTP client) to your local machine *without* retrieving the whole archive file.
(4) HTAR summarizes the work done (time, rate, amount) and then ends.

```
htar -xvf case3/myproject.tar time.txt time2.gif           ---(1A)
   OR
htar -xvf case3/myproject.tar -L tlist                     ---(1B)

  HTAR: Opening HPSS server connection                     ---(2)
  HTAR: Reading index file
  HTAR: Opening archive file

  HTAR: Reading archive file                               ---(3)
  HTAR: x  time.txt, 1085 bytes, 4 media blocks
  HTAR: x  time2.gif, 3452 bytes, 8 media blocks

  HTAR: Extract complete for case3/myproject.tar,          ---(4)
        2 files. total bytes read 116,736 in 0.070 seconds (1.669 MB/s)
  HTAR: HTAR SUCCESSFUL
```

Using CRC Checksums

GOAL: To include Cyclic Redundancy Checksums (CRCs) during a creations (-c) run.

STRATEGY: HTAR provides a command line option (-Hcrc) to cause it to generate a CRC for each member file as it is being copied into the archive file. When this option is specified, HTAR will use additional CPU time, and the I/O performance will noticeably decrease. The actual amount of degradation varies from machine to machine. However, using this option is worthwhile if you are concerned about the reliability of HPSS storage media, such as tapes, and you want to verify that files have not changed when they are read back using HTAR's extract option (-x). In addition, if -Hcrc is specified for a listing run (-t), HTAR will display the member file CRC in square brackets []after the permissions field on each line. If no CRC is available for the member file, an empty pair of brackets is displayed.
To illustrate the difference,
(1) Create an archive file without specifying -Hcrc.
(2) Display its contents (-t), but with -Hcrc specified.
(3) Re-create the archive file, this time with -Hcrc specified during the create operation.
(4) Redisplay its contents with -Hcrc specified.

```
  htar -cvf case3/myproject.tar tim*                           ---(1)

    HTAR: a tim1.txt
    HTAR: a tim2.txt
    HTAR: a tim2a.txt
    HTAR: a tim3.a
    HTAR: a time.txt
    HTAR: a time2.gif
    HTAR: a /tmp/HTAR_CF_CHK_28518_1220351609
    HTAR Create complete for case3/myproject.tar. 399,360 bytes written
         for 6 member files, max threads: 8 Transfer time: 0.082 seconds
         (4.878 MB/s)
         HTAR: HTAR SUCCESSFUL

  htar -Hcrc -tvf case3/myproject.tar                          ---(2)

    HTAR: -rw-r--r-- [] gleicher/hpss 9367 2005-02-04 09:05 tim1.txt
    HTAR: -rw------- [] gleicher/hpss 10568 2008-09-02 03:12 tim2.txt
    HTAR: -rwx------ [] gleicher/hpss 4396 2008-09-02 03:13 tim2a.txt
    HTAR: -rwx------ [] gleicher/hpss 253839 2008-09-02 03:13 tim3.a
    HTAR: -rwx------ [] gleicher/hpss 1786 2008-09-02 03:14 time.txt
    HTAR: -rw------- [] gleicher/hpss 113512 2008-09-02 03:14 time2.gif
    HTAR: -rw------- [] gleicher/gleicher 256 2008-09-02
    03:33 /tmp/HTAR_CF_CHK_28518_1220351609
    HTAR: Listing complete for case3/myproject.tar, 7 files 7 total objects
    HTAR: HTAR SUCCESSFUL

  htar -Hcrc -cvf case3/myproject.tar tim*                     ---(3)

    HTAR: a tim1.txt
    HTAR: a tim2.txt
    HTAR: a tim2a.txt
    HTAR: a tim3.a
    HTAR: a time.txt
    HTAR: a time2.gif
    HTAR: a /tmp/HTAR_CF_CHK_28618_1220351078
    HTAR Create complete for case3/myproject.tar. 399,360 bytes written
    for 6 member files, max threads: 8 Transfer time: 0.066 seconds
    (6.075 MB/s)gleicher@toofast26[/home/toofast/gleicher/temp]:

  htar -Hcrc -tvf case3/myproject.tar                           ---(4)

    HTAR: -rw-r--r-- [0xf4124da1] gleicher/hpss 9367 2005-02-04 09:05 tim1.txt
    HTAR: -rw------- [0x6a20d2c7] gleicher/hpss 10568 2008-09-02 03:12 tim2.txt
    HTAR: -rwx------ [0xaee5f5b9] gleicher/hpss 4396 2008-09-02 03:13 tim2a.txt
    HTAR: -rwx------ [0x036ebadc] gleicher/hpss 253839 2008-09-02 03:13 tim3.a
    HTAR: -rwx------ [0x7acb1840] gleicher/hpss 1786 2008-09-02 03:14 time.txt
    HTAR: -rw------- [0xcc713d8a] gleicher/hpss 113512 2008-09-02 03:14 time2.gif
    HTAR: -rw------- [0xa9bcf0db] gleicher/hpss 256 2008-09-02
    03:24 /tmp/HTAR_CF_CHK_28618_1220351078
    HTAR: Listing complete for case3/myproject.tar, 7 files 7 total objects
    HTAR: HTAR SUCCESSFUL
```

Verify Archive Contents During Creation

GOAL: To verify the contents of the archive file during the creation run.

STRATEGY: HTAR provides the -Hverify=*option[,option...]* command line option, which causes HTAR to first create the archive file normally, and then to go back and check its work by performing a series of checks on the archive file. You choose the types of checks to be performed by specifying one or more comma-separated options. The options can be either individual items, or the keyword all, or a numeric level between 0, 1 or 2. Each numeric level includes all of the checks for lower-valued levels and adds additional checks. The verification options for -Hverify are described in the Control Options (page 25) section.
In the example,
(1) An archive file is created (-c) with verification level 2, including CRC generation and checking. The verbose option (-v) is used to cause HTAR to display information about each file that is added during the create phase and then verified during the verification phase.
(2) The archive file is then listed (-t) using the -Hcrc option to cause HTAR to display the CRC value for each member.

```
htar -cvf case3/myproject.tar -Hcrc -Hverify=2 tim*           ---(1)

  HTAR: a tim1.txt
  HTAR: a tim2.txt
  HTAR: a tim2a.txt
  HTAR: a tim3.a
  HTAR: a time.txt
  HTAR: a time2.gif
  HTAR: a /tmp/HTAR_CF_CHK_28128_1220351451
  HTAR Create complete for case3/myproject.tar. 399,360 bytes written
       for 6 member files, max threads: 7 Transfer time: 0.041 seconds
       (9.857 MB/s)
  HTAR: V tim1.txt, 9367 bytes, 20 media blocks
  HTAR: V tim2.txt, 10568 bytes, 22 media blocks
  HTAR: V tim2a.txt, 4396 bytes, 10 media blocks
  HTAR: V tim3.a, 253839 bytes, 497 media blocks
  HTAR: V time.txt, 1786 bytes, 5 media blocks
  HTAR: V time2.gif, 113512 bytes, 223 media blocks
  HTAR: V /tmp/HTAR_CF_CHK_28128_1220351451, 256 bytes, 2 media blocks
  HTAR: Verify complete. 0 total errors, 7 total objects (7 Files,0 Dirs,0 Hard Links
  HTAR: HTAR SUCCESSFUL

htar -Hcrc -tvf case3/myproject.tar                           ---(2)

  HTAR: -rw-r--r-- [] gleicher/hpss 9367 2005-02-04 09:05 tim1.txt
  HTAR: -rw------- [] gleicher/hpss 10568 2008-09-02 03:12 tim2.txt
  HTAR: -rwx------ [] gleicher/hpss 4396 2008-09-02 03:13 tim2a.txt
  HTAR: -rwx------ [] gleicher/hpss 253839 2008-09-02 03:13 tim3.a
  HTAR: -rwx------ [] gleicher/hpss 1786 2008-09-02 03:14 time.txt
  HTAR: -rw------- [] gleicher/hpss 113512 2008-09-02 03:14 time2.gif
  HTAR: -rw------- [] gleicher/gleicher 256 2008-09-02
  03:33 /tmp/HTAR_CF_CHK_28128_1220351451
```

```
    HTAR: Listing complete for case3/myproject.tar, 7 files 7 total objects
    HTAR: HTAR SUCCESSFUL
```

Rebuilding a Missing Index

GOAL: To rebuild the missing index file for a stored HTAR archive file and thereby (re)enable blocked access to the files within it (and extract some).

STRATEGY: (1) You try to retrieve all files (-xvf) from the HTAR archive myproject.tar in the case3 subdirectory of your storage home directory.
(2) But HTAR cannot find the external index file (here, called myproject.tar.idx) for this archive, and it returns a somewhat cryptic error message, retrieves no requested files, and ends. (File myproject.tar.idx may have been moved, renamed, or accidentally deleted from storage.)
(3) So you execute HTAR again with the special action -X (uppercase, not lowercase, eks) to request rebuilding the external index for the (same) disabled archive.
(4) HTAR opens a preauthenticated connection to your storage (HPSS) home directory, locates the archive in subdirectory case3, scans (but does not retrieve) its contents, and thereby creates a new myproject.tar.idx file (temporarily on local disk, then moved to the same storage directory as the archive file that it supports). HTAR ends.
(5) Now you again try your original (1) file-retrieval request.
(6) HTAR opens a preauthenticated connection to your storage (HPSS) home directory and reports its housekeeping activities (very quickly, in lines that overwrite, so you may not notice all of these status reports on your screen).
(7) HTAR uses its (newly rebuilt) external index to locate the files within the archive and transfers them by parallel connections to your local machine (it transfers all of them because there is no *filelist* on the execute line).
(8) HTAR summarizes the work done (time, rate, amount) and then ends.

```
htar -xvf case3/myproject.tar                             ---(1)

  HTAR: Opening HPSS server connection
  HTAR: Getting HPSS site info
  ERROR: Received unexpected reply from server: 550       ---(2)
  ERROR: Error -1 getting Index File attributes...
  HTAR: HTAR FAILED
  ###WARNING  htar returned non-zero exit status.
              72 = /usr/local/bin/htar.exe...

htar -Xf case3/myproject.tar                              ---(3)

  HTAR: Opening HPSS server connection                    ---(4)
  HTAR: Reading archive
  HTAR: Copying Index File to HPSS... creating file
  HTAR: HTAR SUCCESSFUL

htar -xvf case3/myproject.tar                             ---(5)

  HTAR: Opening HPSS server connection                    ---(6)
```

```
HTAR: Reading index file
HTAR: Opening archive file

HTAR: Reading archive file                                 ---(7)
HTAR: x  tim1.txt, 3503 bytes, 8 media blocks
HTAR: x  tim2.txt, 4310 bytes, 10 media blocks
HTAR: x  tim2a.txt, 5221 bytes, 12 media blocks
HTAR: x  tim3a., 5851 bytes, 13 media blocks
HTAR: x  time.txt, 1085 bytes, 4 media blocks
HTAR: x  time2.gif, 3452 bytes, 8 media blocks

HTAR: Extract complete. total bytes read:                  ---(8)
        28,160 in 0.141 seconds (0.200 MB/s)
HTAR: HTAR SUCCESSFUL
```

Specifying Very Many Files

GOAL: To specify a very large number of input files yet avoid an HTAR command line too long for the shell to accept.

STRATEGY: The input-line-too-long problem is analyzed and explained in the HTAR Limitations (page 17) section above. The best solution is to keep in a separate directory all and only the intended input files, and then specify that directory's name on HTAR's execute line (for recursive processing). But if you failed to take that precaution, you can work around the problem of having too many file names for the shell to accept by using the UNIX FIND utility as shown here.

(1) Run FIND to select the files that you want (here, those whose names begin with T) in a way that processes the file list internally, not by the shell. (The Limitations (page 17) section above explains why FIND's -EXEC option will *not* finish the job here.) Then pipe (|) FIND's output directly into HTAR to build the stored archive that you want (-cf) using the -L option with a hyphen (-) argument to enable standard input (off by default).

(2) Without -v, HTAR shows no verification but does copy its index file to storage when the archive is successfully stored.

```
find . -name 't*' -print | htar -cf test.tar -L -          ---(1)

  HTAR: Opening HPSS server connection
  HTAR: creating HPSS archive file test.tar
  HTAR: Copying index file to HPSS...creating file         ---(2)
  HTAR: HTAR SUCCESSFUL
```

Archiving Between Nonstorage Machines

GOAL: To put files from a directory local to one LC production machine into an HTAR archive file that resides in a directory local to another LC production machine (*not* in HPSS storage as usual).

STRATEGY: Your common home directory is already cross-mounted on every LC production machine (and /nfs/tmp is cross-mounted on many machines), so you never need to use HTAR to transfer home files "between machines." And if you want an HTAR archive deposited on the *same* machine where HTAR runs (instead of in storage), use HTAR's -E option. You only need to follow the steps below if you really need to transfer files from one machine's *local* directory (such as /usr/tmp) to or from an archive file in another machine's *local* directory. Use -F to specify the nonstorage archive host.
(WARNING: using -F to specify "storage" as the archive host is not only completely unnecessary, because it is HTAR's default, but also inefficient, because it undermines HTAR's built-in techniques for moving files to or from *stored* archives quickly and effectively.)

(1) To create (-c) an HTAR archive on another machine (instead of storage), run HTAR on the machine and in the directory where the files to be archived *reside* (because HTAR makes an archive using PUTs). For example:

```
   SOURCE:                         SINK:
YANA                            ATLAS
/var/tmp                        /usr/tmp
file1...n  -----create--->>>    myarchive.tar
[HTAR runs here]                myarchive.tar.idx
```

Use -F to specify the sink machine (where the archive and its index file will go, here ATLAS), and use -f to specify the full pathname of the archive on that sink machine. Note that HTAR still contacts the HPSS server to log this transaction, even though the archive is not stored in HPSS. (Afterwards, you can use -F with -K to verify the remote archive if you are concerned about transfer reliability.)

(2) To extract (-x) file1 from an HTAR archive on another machine (instead of from storage), run HTAR on the machine and in the directory where the files to be extracted should *arrive* (because HTAR uses GETs to extract files). For example:

```
   SINK:                           SOURCE:
YANA                            ATLAS
/var/tmp                        /usr/tmp
file1      <<<---extract----    myarchive.tar
[HTAR runs here]                myarchive.tar.idx
```

Use -F to specify the source machine (where the archive and its index file reside, here ATLAS), and use -f to specify the full pathname of the archive on that source machine. Note that HTAR still contacts the HPSS server to log this transaction, even though the archive is not stored in HPSS.

```
htar -c -f /usr/tmp/myarchive.tar -F atlas *         ---(1)

   HTAR: Opening FTP server connection
   HTAR: Opening HPSS server connection
         Creating FTP archive file /usr/tmp/myarchive.tar
         Pass 2: adjusting local index file entries
         Copying index file to remote host...creating file
   HTAR: HTAR SUCCESSFUL

htar -x -f /usr/tmp/myarchive.tar -F atlas file1     ---(2)

   HTAR: Opening FTP server connection
   HTAR: Opening HPSS server connection
         Reading index file
         Opening archive file
         Reading archive file
   HTAR: HTAR SUCCESSFUL
```

Disclaimer

Keyword Index

To see an alphabetical list of keywords for this document, consult the next section (page 48).

Keyword	Description
entire	This entire document.
title	The name of this document.
scope	Topics covered in this document.
availability	Where HTAR runs.
who	Who to contact for assistance.
introduction	General HTAR overview, analysis.
htar-role	Goals, scope, performance of HTAR.
htar-files	Three key HTAR files diagrammed.
tar-comparison	TAR and HTAR features compared.
htar-usage	How to use HTAR.
execute-line	Required syntax, features, defaults.
htar-errors	Common errors conditions, warnings.
limitations	Known HTAR limitations, work-arounds.
input-output	How to use standard input, output.
environment-variables	Env. variables used by HTAR.
hopper-htar	HOPPER as an HTAR controller.
options	HTAR options grouped, explained.
action	HTAR's action options (1 reqd).
archive	HTAR's archive option (always reqd).
control	HTAR's control options.
examples	Annotated sample HTAR sessions.
create-archive	How to make an HTAR archive.
retrieve-files	How to extract HTAR files.
checksum-use	How to include CRCs during a creation run.
rebuild-index	How to rescue a lost HTAR index.
many-files	Specifying very many input files.
find-input	Specifying very many input files.
between-machines	Archiving between NONstorage machines.
index	The structural index of keywords.
a	The alphabetical index of keywords.
date	The latest changes to this document.
revisions	The complete revision history.

Alphabetical List of Keywords

Keyword	Description
a	The alphabetical index of keywords.
action	HTAR's action options (1 reqd).
archive	HTAR's archive option (always reqd).
availability	Where HTAR runs.
between-machines	Archiving between NONstorage machines.
checksum-use	How to include CRCs during a creation run.
control	HTAR's control options.
create-archive	How to make an HTAR archive.
date	The latest changes to this document.
entire	This entire document.
environment-variables	Env. variables used by HTAR.
examples	Annotated sample HTAR sessions.
execute-line	Required syntax, features, defaults.
find-input	Specifying very many input files.
hopper-htar	HOPPER as an HTAR controller.
htar-errors	Common errors conditions, warnings.
htar-files	Three key HTAR files diagrammed.
htar-role	Goals, scope, performance of HTAR.
htar-usage	How to use HTAR.
index	The structural index of keywords.
input-output	How to use standard input, output.
introduction	General HTAR overview, analysis.
limitations	Known HTAR limitations, work-arounds.
many-files	Specifying very many input files.
options	HTAR options grouped, explained.
rebuild-index	How to rescue a lost HTAR index.
retrieve-files	How to extract HTAR files.
revisions	The complete revision history.
scope	Topics covered in this document.
tar-comparison	TAR and HTAR features compared.
title	The name of this document.
who	Who to contact for assistance.

Date and Revisions

Revision Date	Keyword Affected	Description of Change
26Jan10	control	New -H suboptions
	examples	New example shows use of CRC checksums.
10Sep07	htar-role	Member limit up to 68 Gbyte.
	limitations	Member limit up to 68 Gbyte.
16May06	control	Option -m directory role clarified.
08Aug05	hopper-htar	Drag alternatives added.
12Jul05	introduction	Roles, features expanded.
	execute-line	-P role noted with -c.
	environment-variables	TMPDIR role added.
	hopper-htar	New section on HTAR controller.
	control	Options -n, -P, -q added.
	create-archive	-P added to example.
	retrieve-files	HOPPER added as alternative.
	index	New keywords added.
08Sep04	htar-role	Parallel connections clarified (not PFTP).
	tar-comparison	-L now recursive.
	execute-line	Syntax clarified, -K added.
	htar-errors	Error labels spelled out.
	limitations	Standard input (-L), output (-O) enabled.
	environment-variables	HPSS.conf role explained.
	action	Option -K added.
	control	New details on -F, -L, -M, -T. New -O option added.
	examples	Using -F, -L details updated.
	index	New keyword added.
17Nov03	htar-role	Size and speed details updated.
	limitations	Size and speed details updated.
	execute-line	General transfers with -F clarified.
	action	General transfers with -F noted.
23Jul03	between-machines	New section on nonstorage archiving.
	index	New keyword for section.
	examples	More verbose, explicit HTAR output.
	tar-comparison	Details updated for new features.
	limitations	Only passwordless FTP servers allowed.
	control	-F, -H, -M control options added.
	introduction	Class of service clarified.
13May03	availability	HTAR now runs under Linux/CHAOS.

	htar-role	Use NETMON for performance data.
	execute-line	Consistency file location under Linux.
24Jun02	htar-role	Size and copy issues clarified.
	execute-line	Y option added.
		Single-copy default now, dual with Y.
	limitations	Maximum file size clarified.
	control	-Y 150 for dual copy noted.
01May02	htar-errors	Retrieval with filters problem noted.
	limitations	Retrieval with filters problem noted.
	control	-L use expanded.
	retrieve-files	
		Example now treats filter problem.
17Oct01	options	-S and -h roles clarified.
		Options section subdivided.
	index	New keywords for subsections.
20Aug01	entire	First edition of HTAR manual.

EJG (26Jan10)